

Distributed development tools and management approaches for the VxO environment: The Linux lessons

Who? Jan Merka¹ A. Szabo² T. Narock¹ T. King³ R. Walker³

From? ¹GEST, University of Maryland Baltimore County, Baltimore, MD (jan.merka@gsfc.nasa.gov)

²NASA Goddard Space Flight Center, Greenbelt, MD

³IGPP, University of California, Los Angeles, CA

When? Virtual Observatories in Geosciences, Denver, CO
June 12–14, 2007

Abstract

Each NASA Virtual Observatory (VxO) is a distributed data environment that communicates or will eventually communicate with other VxOs and the entire VxO environment will share some resources of which the most important one is the metadata. Creation of SPASE data descriptions is a wide-spread effort that is akin to Open Source Software (OSS) development. Complex OSS projects, e.g. the Linux kernel, rely on the network of trust, where the lead developers trust a limited number of skilled developers, those in turn have their own trusted collaborators and so on. It is apparent that the VxO development is in many ways similar and the VxO teams should consider employing tools and/or processes that facilitate efficient development. As a particular example, we will focus the presentation on discussing metadata exchange, creation and the need for using a revision management system. We will argue that distributed repositories, e.g. git, are more suitable for VxOs than centralized systems (CVS, Subversion, ...).

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

VxO Environment

- Distributed data sources
- Distributed development (metadata, tools, ...)
- VxOs are not co-located
- Did I say **distributed** again?

Question #1 How do we manage the development?

Question #2 What kind of tools can help us?

Question #3 To create or not to create the tools?

Answer The VxO development is similar to Open Source Software (OSS) projects and we should study and employ their management methods and development tools.

The Metadata Example

- Resource descriptions (observatory, instrument, data product, ...) are **created and shared** by various persons or groups across VxOs
- Metadata exchange by email or direct download
- Difficult to keep track of versions and content changes

Wait! What about metadata registries and registry servers? Can they help?

Eeeeeee, NO! Registries or registry servers are not a solution unless they provide Source Code (Content) Management (SCM) features! Their purpose is to find and deliver **existing** metadata, not to provide assistance in metadata development.

The Metadata Example

- Resource descriptions (observatory, instrument, data product, ...) are **created and shared** by various persons or groups across VxOs
- Metadata exchange by email or direct download
- Difficult to keep track of versions and content changes

Wait! What about metadata registries and registry servers? Can they help?

Eeeeeee, NO! Registries or registry servers are not a solution unless they provide Source Code (Content) Management (SCM) features! Their purpose is to find and deliver **existing** metadata, not to provide assistance in metadata development.

The Metadata Example

- Resource descriptions (observatory, instrument, data product, ...) are **created and shared** by various persons or groups across VxOs
- Metadata exchange by email or direct download
- Difficult to keep track of versions and content changes

Wait! What about metadata registries and registry servers? Can they help?

Eeeeeee, NO! Registries or registry servers are not a solution unless they provide Source Code (Content) Management (SCM) features! Their purpose is to find and deliver **existing** metadata, not to provide assistance in metadata development.

The Metadata Example

- Resource descriptions (observatory, instrument, data product, ...) are **created and shared** by various persons or groups across VxOs
- Metadata exchange by email or direct download
- Difficult to keep track of versions and content changes

Wait! What about metadata registries and registry servers? Can they help?

Eeeeeee, NO! Registries or registry servers are not a solution unless they provide Source Code (Content) Management (SCM) features! Their purpose is to find and deliver **existing** metadata, not to provide assistance in metadata development.

The Metadata Example

- Resource descriptions (observatory, instrument, data product, ...) are **created and shared** by various persons or groups across VxOs
- Metadata exchange by email or direct download
- Difficult to keep track of versions and content changes

Wait! What about metadata registries and registry servers? Can they help?

Eeeeeee, NO! Registries or registry servers are not a solution unless they provide Source Code (Content) Management (SCM) features! Their purpose is to find and deliver **existing** metadata, not to provide assistance in metadata development.

OSS Development Approach

Definition Unexpected things^(TM) happen!

The approach

- Keep the work (source code, documentation, ...) in a content management system (SCM)
- Public access to the code and documentation — encourage usage and testing
- Communication via mailing lists.
- Trusted developers update code

Example The most famous OSS project, Linux, employs a distributed SCM system *git* to keep track of the content of its ~22,000 files.

OSS Development Approach

Definition Unexpected things^(TM) happen!

The approach

- Keep the work (source code, documentation, ...) in a content management system (SCM)
- Public access to the code and documentation — encourage usage and testing
- Communication via mailing lists.
- Trusted developers update code

Example The most famous OSS project, Linux, employs a distributed SCM system *git* to keep track of the content of its ~22,000 files.

OSS Development Approach

Definition Unexpected things^(TM) happen!

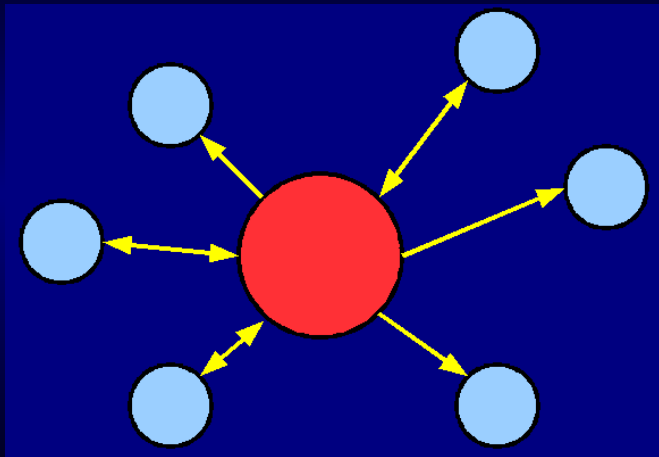
The approach

- Keep the work (source code, documentation, ...) in a content management system (SCM)
- Public access to the code and documentation — encourage usage and testing
- Communication via mailing lists.
- Trusted developers update code

Example The most famous OSS project, Linux, employs a distributed SCM system *git* to keep track of the content of its ~22,000 files.

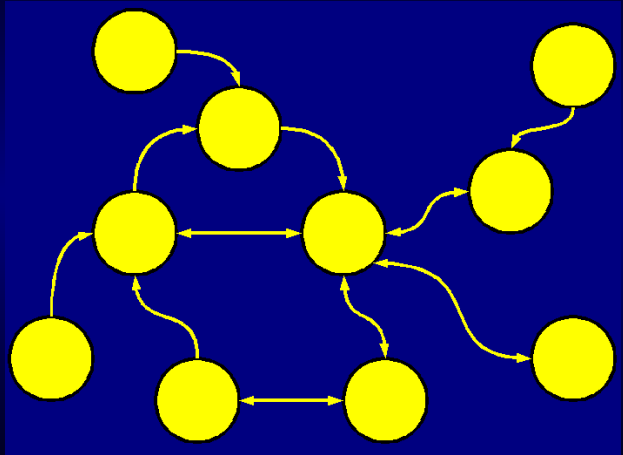
Centralized SCM

- CVS, Subversion (SVN), Perforce, ...



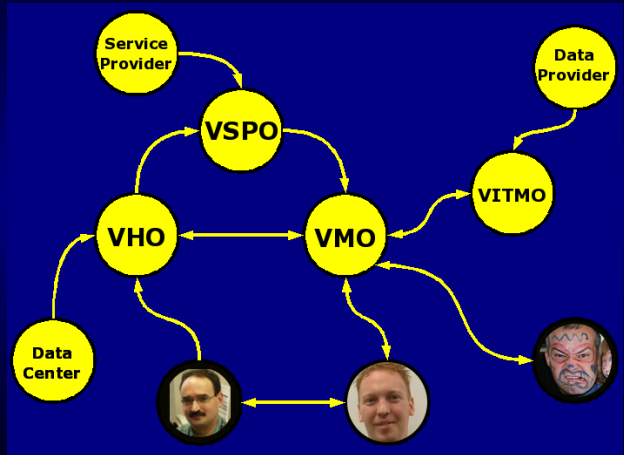
Distributed SCM

- git, BitKeeper, Mercurial, bazaar-ng, ...



Distributed SCM

- git, BitKeeper, Mercurial, bazaar-ng, ...






Why Does Linus Torvald Use *git* for Linux?¹

Answer *Because he created git.*

Answer *Because git is better than any other SCM.*

- Reliable
- High performance
- Distributed
- Content manager




¹Linus Torvald on git: <http://www.youtube.com/watch?v=4XpnKHJAok8>   

Why Does Linus Torvald Use *git* for Linux?¹

Answer *Because he created git.*

Answer *Because git is better than any other SCM.*

- Reliable
- High performance
- Distributed
- Content manager

¹Linus Torvald on git: <http://www.youtube.com/watch?v=4XpnKHJAok8>   

Why Does Linus Torvald Use *git* for Linux?¹

Answer *Because he created git.*

Answer *Because git is better than any other SCM.*

- Reliable
- High performance
- Distributed
- Content manager

¹Linus Torvald on git: <http://www.youtube.com/watch?v=4XpnKHJAok8>

Distribution

Distributed

- Everybody has his own branch (or several), branching is inherent
- Branching and especially merging in centralized SCMs is difficult so people don't do it (much).
- Commits are local
- Pull from each other within a (sub)group and merge within a (sub)group. When finished, ask the main group to pull from them.

Collaboration

- Commits do not disturb others
- No special *write access* allows for much higher trust and security

Off-line work

- Network connection is not needed to work with the repository (logs, commits, ...), so you can work anywhere (without access to the server).

Distribution

Distributed

- Everybody has his own branch (or several), branching is inherent
- Branching and especially merging in centralized SCMs is difficult so people don't do it (much).
- Commits are local
- Pull from each other within a (sub)group and merge within a (sub)group. When finished, ask the main group to pull from them.

Collaboration

- Commits do not disturb others
- No special *write access* allows for much higher trust and security

Off-line work

- Network connection is not needed to work with the repository (logs, commits, ...), so you can work anywhere (without access to the server).

Distribution

Distributed

- Everybody has his own branch (or several), branching is inherent
- Branching and especially merging in centralized SCMs is difficult so people don't do it (much).
- Commits are local
- Pull from each other within a (sub)group and merge within a (sub)group. When finished, ask the main group to pull from them.

Collaboration

- Commits do not disturb others
- No special *write access* allows for much higher trust and security

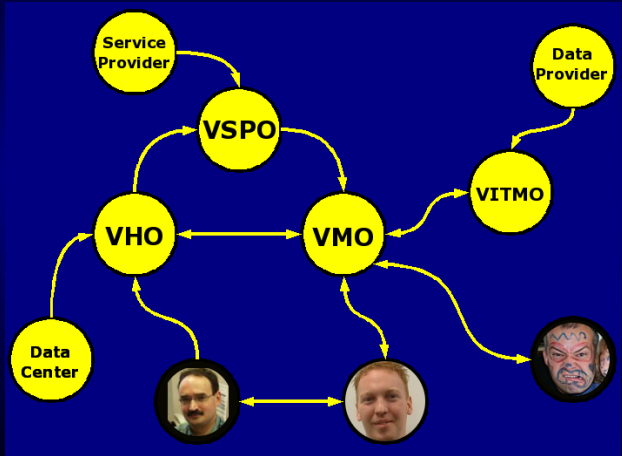
Off-line work

- Network connection is not needed to work with the repository (logs, commits, ...), so you can work anywhere (without access to the server).

Network of Trust

Which branch to use?

Use only branches from trusted sources. In practice, only a few branches are widely used.



Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at *every* use

Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at *every* use

Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at *every* use

Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at *every* use

Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at *every* use

Reliable: Trust & Security

Reliable SCM guarantees that you get what you put in.

Distributed systems are mostly inherently safer:

- No single point of failure - no single repository is more important than other
- Natural replication of data - you can recover your repository by pulling someones branch
- Natural security boundaries - people can make changes without write access to your computer!
- Automatic corruption check - git checksums everything with a strong hash (SHA1) and checks it at every use

Performance

If a tool can do something really fast and well, people will start using it!

Linux Torvald on git

- Performance affects how you work and affects quality
- Merge of the kernel (22,000 files) takes git less than a second
- Git enables easy merging, in contrast to centralized SCMs, and this encourages developers to merge often and early

Content Management

Repository is much more than just a random collection of files.

- Git tracks content, not files:

Example

What happened to [part of] the project?
(Project=SPASE descriptions, Part=VMO-guaranteed metadata)

- Git can tell history of a function moved from one file to another.
no other SCM can do that. But git doesn't tell *file* history.
- History must always be seen on a project basis!

Content Management

Repository is much more than just a random collection of files.

- Git tracks content, not files:

Example

What happened to [part of] the project?
(Project=SPASE descriptions, Part=VMO-guaranteed metadata)

- Git can tell history of a function moved from one file to another.
no other SCM can do that. But git doesn't tell *file* history.
- History must always be seen on a project basis!

Content Management

Repository is much more than just a random collection of files.

- Git tracks content, not files:

Example

What happened to [part of] the project?
(Project=SPASE descriptions, Part=VMO-guaranteed metadata)

- Git can tell history of a function moved from one file to another.
no other SCM can do that. But git doesn't tell *file* history.
- History must always be seen on a project basis!

Content Management

Repository is much more than just a random collection of files.

- Git tracks content, not files:

Example

What happened to [part of] the project?
(Project=SPASE descriptions, Part=VMO-guaranteed metadata)

- Git can tell history of a function moved from one file to another.
no other SCM can do that. But git doesn't tell *file* history.
- History must always be seen on a project basis!

Content Management

Repository is much more than just a random collection of files.

- Git tracks content, not files:

Example

What happened to [part of] the project?
(Project=SPASE descriptions, Part=VMO-guaranteed metadata)

- Git can tell history of a function moved from one file to another.
no other SCM can do that. But git doesn't tell *file* history.
- History must always be seen on a project basis!

The Lesson

VxOs

- Face similar problems as Linux kernel development (distributed, distributed, distributed, . . . , distributed)
- Need to keep track of and exchange metadata and source code — they should employ SCMs (and not develop their own equivalent tools)
- Should consider a distributed SCM like *git* which has proven superior over other SCMs in Linux kernel development